

## Лабораторная работа

### Реализация простейшего алгоритма распознавания графических образов

Рассматривается программа распознавания рукописных прописных русских букв и цифр на основе метода сравнения с эталонными изображениями соответствующих символов.

Данный подход может быть использован для написания собственных модулей распознавания символов (в том числе рукописных) в разрабатываемом прикладном ПО.

Ниже приведены основные моменты реализации предлагаемого алгоритма.

#### Шаг 1. Создание канвы для рисования и формирование ее образа в памяти.

В качестве канвы используем класс TBitmap (для простоты работы с битмапом используем режим 1 байт на пиксель, т.е. TBitmap.PixelFormat := pf8bit), визуализируем его на TPaintBox, отображаем в памяти при помощи структуры:

```
type
  MasX = PByteArray;
var
  MasY : array of MasX // массив пикселей,
  {
    где MasY[y-коорд][x-коорд] = номер цвета в палитре цветов (при 8 бит/пиксель).
    Отображение осуществляем с использованием TBitmap.ScanLine (быстро и просто):
    SetLength(MasY, TBitmap.Height);
    for j := 0 to TBitmap.Height - 1 do
      MasY[j] := TBitmap.ScanLine[j];
  }
```

Теперь с картинкой в виде матрицы XxY можно делать все что угодно...

#### Шаг 2. Формирование массива эталонных образцов символов.

Эталонные образцы будем формировать на основе матрицы размером 16x16. Для этого разработаем процедуру генерации такой матрицы по произвольному изображению эталона.

Процедура **function Create\_16x16(Img : TBitmap) : TMas16x16** получает в качестве параметра ссылку на картинку, на которой нарисован эталон символа (в нашем случае - программно), возвращает приведенную матрицу размером 16x16.

Кратко поясним работу процедуры (более полно см. комментарии в программе).

1. Получаем ссылку на битмап и осуществляем его отображение в памяти (см. выше).
2. Вычисляем координаты границ (описанного прямоугольника) образа (эталонного или распознаваемого) путем сканирования строк/столбцов. При этом здесь и при дальнейшем анализе изображения предполагается, что символ нарисован черным цветом (№0 в палитре цветов) и соответственно все значащие пиксели имеют значение 0.

```
for j := 0 to Img.Height - 1 do           // Top
begin
  for i := 0 to Img.Width - 1 do
    if MasY[j][i] = 0 then
      begin yTop := j; break; end;
    if yTop = j then break;
  end;

  for j := Img.Height - 1 downto 0 do      // Bottom
begin
  for i := 0 to Img.Width - 1 do
    if MasY[j][i] = 0 then
      begin yBottom := j + 1; break; end;
    if yBottom = j + 1 then break;
  end;

  for i := 0 to Img.Width - 1 do          // Left
begin
  for j := 0 to Img.Height - 1 do
    if MasY[j][i] = 0 then begin xLeft := i; break; end;
```

```

    if xLeft = i then break;
end;

for i := Img.Width - 1 downto 0 do           // Right
begin
    for j := 0 to Img.Height - 1 do
        if MasY[j][i] = 0 then begin xRight := i + 1; break; end;
        if xRight = i + 1 then break;
    end;
end;

```

3. Для дальнейшего анализа потребуется некий критерий, по которому будет производиться свертка исходного изображения символа в матрицу 16x16. Таким критерием был выбран общий процент заполнения - отношение количества значимых пикселей (из которых состоит символ) к общему количеству пикселей в описанном вокруг исходного изображения прямоугольнике. Данный параметр может влиять на качество распознавания, причем если он больше 1 для распознаваемого символа будет соответствовать меньшее количество возможных альтернатив, при значении меньшем 1 - наоборот. В нашем случае коэффициент поправки принят равным 0,99.

```

nSymbol := 0;
for j := yTop to yBottom do
for i := xLeft to xRight do
    if MasY[j][i] = 0 then inc(nSymbol);
Percent := nSymbol / ((yBottom - yTop)*(xRight - xLeft));
Percent := 0.99*Percent;

```

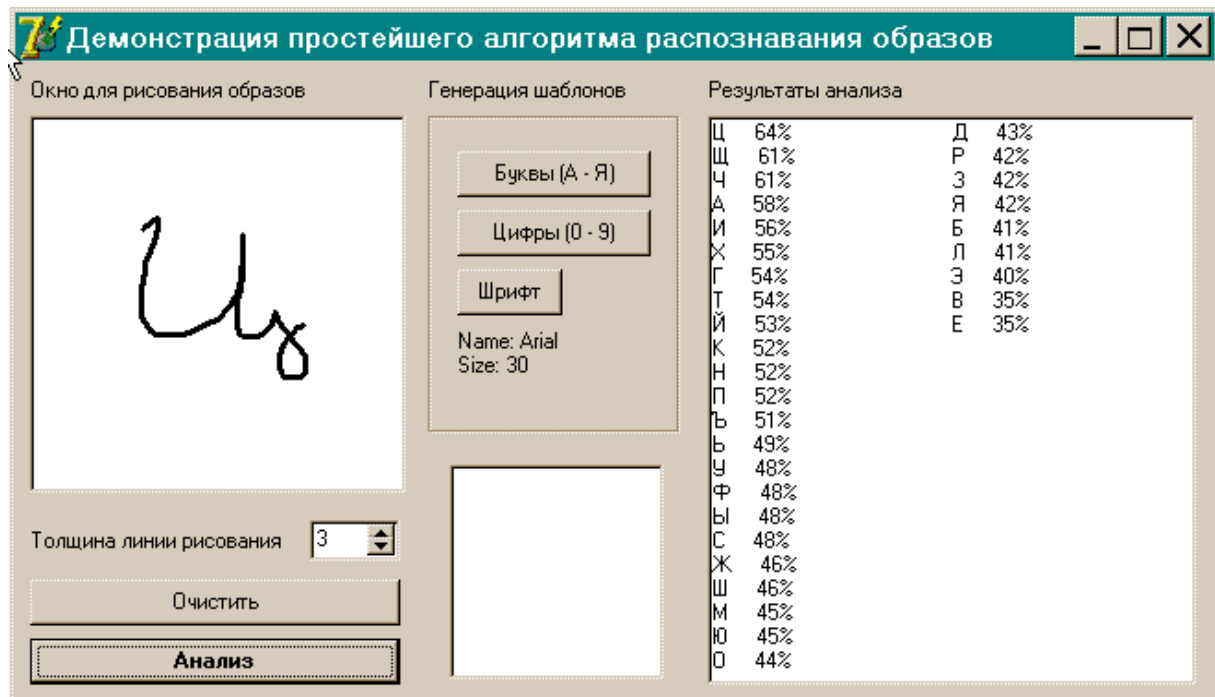
4. Далее разбиваем прямоугольник с изображением символа на 16x16 ячеек путем деления сторон новой ячейки на 2. Запоминаем относительные координаты каждой ячейки и приступаем к заполнению матрицы 16x16. Принимаем в качестве критерия общий процент заполнения. Если в анализируемой ячейке процент заполнения больше, чем общий процент - соответствующий элемент матрицы 16x16 устанавливается в 1, в противном случае - в 0.
5. Остальная часть алгоритма касается вопросов рисования на TBitmap букв или цифр (в цикле), запоминания в массиве матриц 16x16, соответствующих каждому эталонному символу (см. приведенный код).

### Шаг 3. Распознавание рисованных (от руки) символов.

Распознавание осуществляем путем сравнения матрицы 16x16 распознаваемого символа с матрицей эталона (путем перебора имеющихся в наличии). Сравнение производим поэлементно при помощи оператора XOR. Результат - матрица 16x16, содержащая единицы в местах несовпадений тест-символа и эталона. Путем подсчета количества несовпадений формируем вектор, содержащий эту информацию для каждого эталонного символа, и производим сортировку эго элементов по возрастанию количества несовпадений.

Параметр  $(1 - \text{Result}[i]/256)*100\%$ , где **Result[i]** - кол-во несовпадений для **i** - го символа, показывает "вероятность" соответствия образа конкретному символу.

Демонстрационная программа.



1. Сгенерируйте массив шаблонов для букв или цифр, используя конкретный шрифт
2. Нарисуйте от руки произвольную букву (русскую, прописную) или цифру
3. Нажмите на кнопку анализ
4. Исследуйте результат
5. Очистите окно и повторите пп. 2-4.

Что дальше?

Данный алгоритм как простейший обладает рядом существенных ограничений.

Для повышения точности распознавания отдельных символов (не слов, - это другая задача, в каком-то смысле более простая), необходимо проводить дополнительный анализ значимых признаков, например симметричность образа (горизонтальная, вертикальная), наличие замкнутых областей (О, В, Д, Р и др.), количество отрезков и дуг, их взаимное расположение и ориентация (требуется векторизация изображения).

### Задание на лабораторную работу:

- Доработать программу, что бы она могла распознавать 2 буквы (слова).